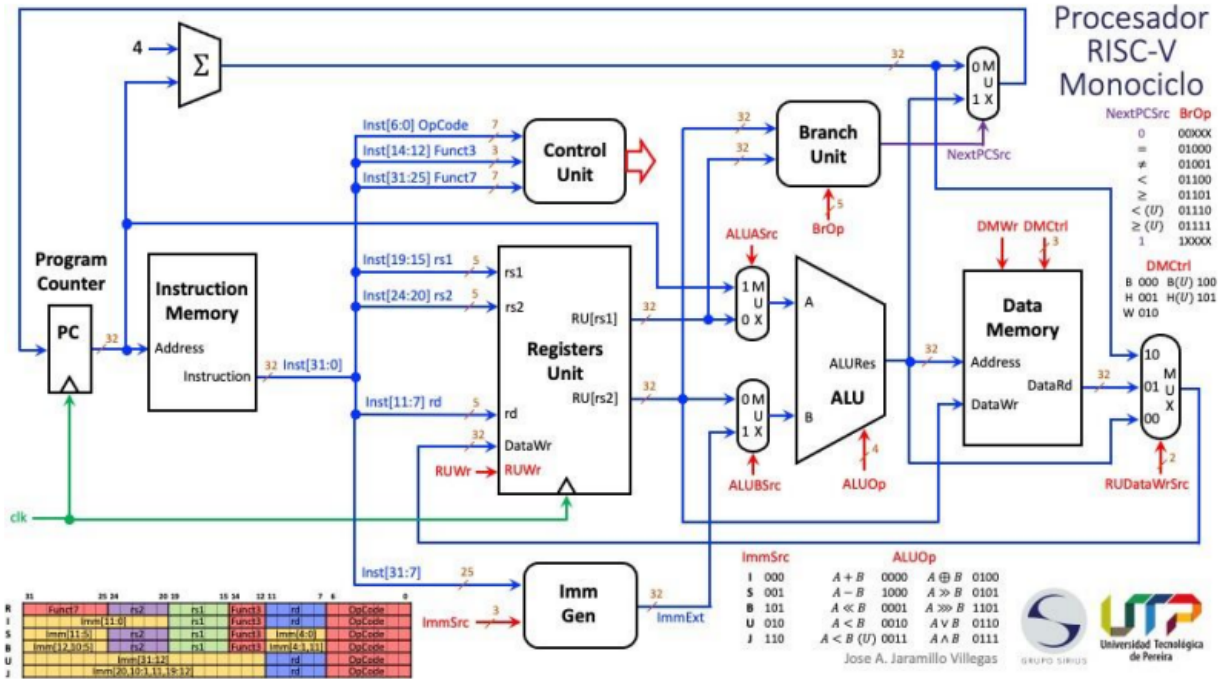
Laboratotio Arquitectura de Computadoras

Santiago Ramirez Arenas
Docente: José Alfredo Jaramillo Villegas
Universidad Tecnológica de Pereira

27 de noviembre de 2021

Procesador monociclo.

Este tipo de procesador tiene como principio que cada instrucción debe ser ejecutada en un ciclo del reloj. Cuenta con varios módulos que logran darle la capacidad de procesamiento los cuales están representados en el gráfico.



Diseño Proceador Monociclo

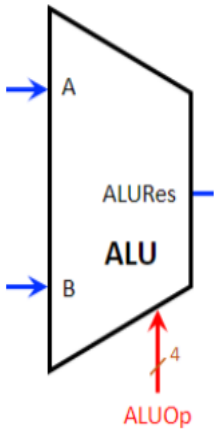
```
'include "instruction_memory.sv"
'include "registers_unit.sv"
'include "ALU.sv"
'include "data_memory.sv"
'include "imm_gen.sv"
'include "branch_unit.sv"
'include "control_unit.sv"
'include "program_counter.sv"
'include "adder.sv"
'include "MUX2x1.sv"
'include "MUX3x1.sv"

module procesador_monociclo (input CLK);
  wire CLK;
  wire [31:0] PC_Out;
  wire [31:0] adder_Out;
  wire [31:0] Instruction;
  wire [1:0] RUDataWrSrc;
  wire ALUASrc, ALUBSrc;
  wire [31:0] Data1;
  wire [31:0] Data2;
  wire RUWr;
  wire [2:0] ImmSrc;
  wire [31:0]ImmExt;
  wire [4:0] BrOp;
  wire  NextPCSrc;
  wire [31:0] ALURes;
  wire [3:0]ALUOp;
  wire [31:0] DataRd;
  wire DMWr;
  wire [2:0] DMCtrl;
  wire [31:0] Out_mux3x1;
  wire [31:0] Out_pc_mux;
  wire [31:0] ALUMUXResA;
  wire [31:0] ALUMUXResB;

  program_counter pc (
          .CLK (CLK,)
          .PC_IN (out_pc_mux),
          .PC_Out (PC_Out));

  adder add (
           .adder_In (PC_Out),
           .adder_Out (adder_Out));

  instruction_memory im (
            .Addrress (PC_out),
            . Instruction (Instruction));

  control_unit cu (
    .OpCode (Instruction [6:0]),
             .Funct3 (Instruction [14:12]),
```

```
            .Funct7 (Instruction [31:25]),
            .ALUASrc (ALUASrc),
            .ALUBSrc (ALUABrc),
            .ALUOp (ALUOp),
            .ImmSrc (ImmSrc),
            .DMWr (DMWr),
            .DMCtrl (DMCtrl),
            .RUDataWrSrc (RUDataWrSrc),
            .RUWr (RUWr),
            .BrOp (BrOp));

 registers_unit ru (
            .rs1 (Instruction [19:15]),
            .rs2 (Instruction[24:20]),
            .rd (Instruction [11:7]),
            .DataWr (Out_mux3x1),
            .RUWr (RUWr),
            .CLK (CLK),
            .Data1 (Data1),
            .Data2 (Data2));

 imm_gen ig (
            .Inst (Instruction),
            .ImmSrc (ImmSrc),
            .ImmExt (ImmExt));

 branch_unit bu (
            .A (Data1),
            .B (Data2),
            .BrOp (BrOp),
            .NextPCSrc (NextPCSrc));

 MUX2x1 alu_mux_a (
            .In1 (Data1),
            .In2 (PC_Out),
            .Src (ALUASrc),
            .Out (ALUMUXResA));

MUX2x1 alu_mux_b (
            .In1 (Data2),
            .In2 (ImmExt),
            .Src (ALUBSrc),
            .Out (ALUMUXResB));

 ALU alu (
            .A(ALUMUXResA),
            .B(ALUMUXResB),
            .ALUOp (ALUOp),
            .ALURes (ALURes));

 data_memory dm (
            .Address (ALURes),
            .DataWr (Data2),
            .DMWr (DMWr),
```

```
107             .DMCtrl (DMCtrl),
108             .DataRd (DataRd));
109
110   MUX3x1 mux3to1(
111             .In1 (ALURes),
112             .In2 (DataRd),
113             .In3 (adder_Out),
114             .RUDataWrSrc (RUDataWeSrc),
115             .Out (Out_mux3x1));
116
117   MUX2x1 pc_mux(
118     .In1 (adder_Out),
119     .In2 (ALURes),
120     .Src (NextPCSrc),
121     .Out (Out_pc_mux));
122 endmodule
```

Unidad aritmeticológica (ALU):

Creación de una ALU de dos operandos (A y B tamaño de 32 bits), un ALUOp (tamaño de 4 bits) que asigna una operación y devuelve el resultado (tamaño de 32 bits).

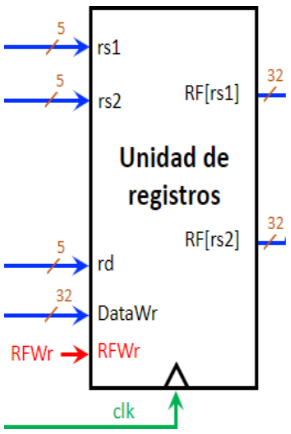



```
1  module ALU (
2    input logic signed [31:0]A,
3    input logic signed [31:0]B,
4    input logic [3:0]ALUOp,
5    output logic signed [31:0]ALURes);
6
7    always @ (*)
8           begin
9             case (ALUOp)
10               4'b0000:
11                   ALURes <= A + B;
12               4'b1000:
13                   ALURes <= A - B;
14               4'b0001:
15                   ALURes <= A << B;
16               4'b0010:
```

```
                    ALURes <= A < B;
                4'b0011:
                    ALURes <= $unsigned (A);
                4'b0100:
                    ALURes <= A ^ B;
                4'b0101:
                    ALURes <= A >> B;
                4'b1101:
                    ALURes <= A >>> B;
                4'b0110:
                    ALURes <= A | B;
                4'b0111:
                    ALURes <= A & B;
                4'b1111:
                    ALURes <= A <= B;
            endcase
        end
endmodule
```

- Unidad de registros:

Creación del módulo Unidad de Registros. El cual cuenta con los siguientes registros (cada uno con su respectivo número de bits): el ingreso de la dirección de dos registros(rs1, rs2), dos salidas con la información que fue solicitada (RF[rs1], RF[rs2]), activador de lectura (rd), activador de escritura(RFWr) e información de registro para ingresar(DataWrz).




```
module registers_unit (
  input logic  [4:0]rs1,
  input logic  [4:0] rs2,
  input logic  [4:0] rd,
  input logic  [31:0] DataWr,
  input logic  RUWr,
  input logic CLK,
  output logic [31:0] Data1,
  output logic [31:0] Data2);

  logic [31:0] RU [31:0];
```

```
12
13        initial
14          $readmemb ("registers.txt",RU);
15
16    assign Data1 = RU [rs1];
17    assign Data2 = RU [rs2];
18
19    always @ (posedge CLK)
20          begin
21            if (RUWr == 1 && rd != 5'b0)
22              RU [rd] <= DataWr;
23             $monitor ("x20 value: %d", RU [20])
24          end
25  endmodule
```

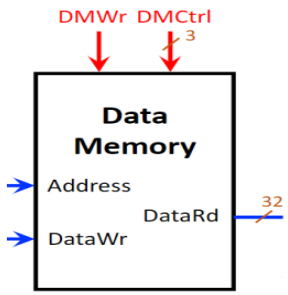
Memoria de datos.

Se define como una matriz en la cual se puede almacenar y leer los datos que van a ser procesados. Este módulo del procesador se encarga de almacenar todos los datos que van a perdurar y, a su vez, recuperar los datos solicitados. El módulo cuenta con los siguientes registros de entrada y salida (cada uno con su respectivo número de bits):

Escritura: La dirección de ingreso para almacenar (Address), el valor que se quierealmacenar (DataWr) y la bandera que activa la escritura (DMWr).

Lectura: Tipo de lectura DMCtrl (Byte, Halfword, Word, Unsigned Byte, Unsigned Halfword), La dirección de ingreso para leer la memoria(Address).

Nota: Address funciona para los casos (lectura y escritura)

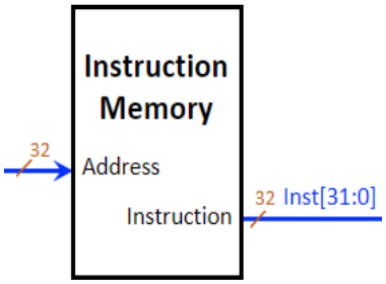



```
1  module data_memory (
2    input logic [31:0] Address,
3    input logic signed [31:0] DataWr,
4    input logic DMWr,
5    input logic [2:0] DMCtrl,
6    output logic signed [31:0] DataRd );
7
8         integer 1;
9         parameter memory_size 2**20;
10        logic [7:0] Memory [Memory_size -1: 0];
```

```
11
12      //Escritura
13        always @ (*)
14        begin
15          if (DMWr == 1'b1)
16                 begin
17                   case (DMCrel)
18                          3'b000: // Byte
19                            Memory [Address] <= DataWr [7:0];
20                          3'b001: // Media palabra
21                                 begin
22                            Memory [Address] <= DataWr [7:0];
23                            Memory [Address + 1] <= DataWr [15:8];
24                                 end
25                           3'b010: // Palabra
26                                 begin
27                                   for (i = 0; 1 < 4; i = i + 1)
28                         Memory [Address + 1] <= DataWr [i*8 + :8];
29                                 end
30                          endcase
31                    end
32                end
33
34          //Lectura
35             always @ (*)
36             begin
37               case (DMCtrl)
38     3'b000: // Byte
39         DataRd <= {{24{Memory[Address][7]}}, Memory [Address]};
40     3'b001: // Media palabra
41         DataRd <= {{16{Memory[Address + 1][7]}}, Memory [Address + 1],
    Memory [Address]};
42     3'b010: // Palabra
43         DataRd <= {{Memory[Address + 3], Memory [Address + 2], Memory
    [Address + 1], Memory[Address]};
44     3'b100://Byte unsigned
45         DataRd <= {24'b0, Memory [Address]};
46     3'b101 // Media palabra unsigned
47         DataRd <= {16'b0, Memory [Address + 1], Memory[Address]};
48                               default:
49                              DataRd = 31'bx;
50                    endcase
51                end
52            endmodule
```

Memoria de instrucciones.

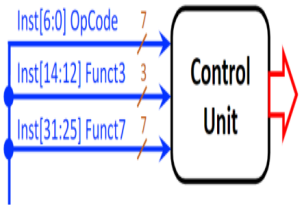
Se puede definir como una matriz en la cual se almacenan y se leen las instrucciones que se van a ejecutar. Este módulo del procesador recibe un Address el cual es la dirección de la instrucción que se quiere recuperar y una salida Instruction la cual es la instrucción que fue recuperada de la matriz




```
module intruction_memory (
  input logic [31:0]Address,
  input logic [31:0]Instruction);

       parameter memory_ize = 2 **20;

  logic [7:0] Memory [memory_size - 1: 0];

  initial begin
    $readmemh ("instructions.txt", Memory);
  end

  always @ (*)
    Instruction <={Memory [Address + 3], Memory[Address + 2],
   Memory[Address + 1], Memory [Address]};
endmodule
```

Unidad de control

Es la encargada de decodificar la instrucción de entrada (OpCode, Funct3, Funct7) para generar cada una de las señales de control (RUWr, ALUASrc, ALUBSrc, ALUOp, BrOp, DMWr, DMCtrl, RUDataWrSrc e ImmSrc).



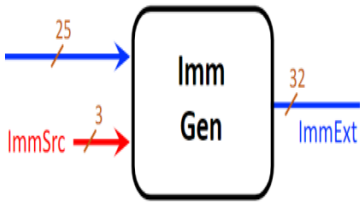
```
module control_unit (
  input logic [6:0].OpCode,
  input logic [2:0] Funct3,
  input logic [6:0] Funct7,
  output logic ALUASrc,
  output logic ALUBSrc,
  output logic [3:0] ALUOp,
  output logic [2:0] ImmSrc,
  output logic DMWr,
  output logic [2:0] DMCtrl,
  output logic [1:0] RUDataWeSrc,
  output logic  RUWr,
  output logic  [4:0]BrOp;

  always @ (*)
         begin
           case (OpCode)
                  7'b0110011; //Instrucion tipo R
                  begin
                         ALUASrc = 1'b0;
                         ALUBSrc = 1'b0;
                    ALUOp = {Funct7 [7], Funct3};
                    ImmSrc = 3'bxxx;
                    DMWr = 1'b0;
                    CMCtrl = 3'bxxx;
                    RUDataWrSrc = 2'b00;
                    RUWr = 1'b1;
                    BrOp = 5' b00xx;
                  end

                  7'b0010011: // Intruccion tipo I
                 begin
                         ALUASrc = 1'b0;
                         ALUBSrc = 1'b1;
                         if (Funct3 == 3'b101)
                            ALUOp = {Funct7 [5], Funct3};
                         else
                           ALUOp = {1'b0, Funct3};
                           ImmSrc = 3'b000;
                           DMWr = 1'b0;
                           DMCtrl = 3'bxxx;
                           RUDataWeSrc = 2'b00;
                           RUWr = 1'b1;
                           BrOp = 5'b00xx;
                  end

                  7'b0000011: // Instruccion tipo I load
                  begin
                          ALUASrc = 1'b0;
                          ALUBSrc = 1'b1;
                          ALUOp = 4'b0000;
                          ImmSrc = 3'b000;
                          DMWr = 1'b0;
                          DMCtrl = Funct3;
```

```
                        RUDataWrSrc = 2'b01;
                        RUWr = 1'b1;
                        BrOp = 5'b00xxx;
                end

                7'b1100111: // Instruccion tipo I jalr
                begin
                        ALUASrc = 1'b0;
                        ALUBSrc = 1'b1;
                        ALUOp = 4'b0000;
                        ImmSrc = 3'b000;
                        DMWr = 1'b0;
                        DMCtrl = 3'bxxx;
                        RUDataWrSrc = 2'b10;
                        RUWr = 1'b0;
                        BrOp = 5'b1xxxx;
                end

                7'b1100011: //Instruccion tipo B
                begin
                        ALUASrc = 1'b1;
                        ALUBSrc = 1'b1;
                        ALUOp = 4'b0000;
                        ImmSrc = 3'b101;
                        DMWr = 1'b0;
                        DMCtrl = 3'bxxx;
                        RUDataWrSrc = 2'bxx;
                        RUWr = 1'b0;
                        BrOp = {2'b01, Funct3};
                end

                 7'b0100011: //Instruccion tipo S
                begin
                        ALUASrc = 0;
                        ALUBSrc = 1;
                        ALUOp = 4'b0000;
                        ImmSrc = 3'b001;
                        DMWr = 1'b1;
                        DMCtrl = Funct3;
                        RUDataWrSrc = 2'bxx;
                        RUWr = 1'b0;
                        BrOp = 5'b00xx;
                end

                7'b1101111: //Instruccion tipo J -Jal
                begin
                        ALUASrc = 1'b1;
                        ALUBSrc = 1'b1;
                        ALUOp = 4'b0000;
                        ImmSrc = 3'b110;
                        DMWr = 1'b0;
                        DMCtrl = 3'bxxx;
                        RUDataWrSrc = 2'b10;
                        RUWr = 1'b1;
```

```
                        BrOp = 5'b1xxxx;
                end

                 7'b0110111: //Instruccion tipo U -lui
                begin
                        ALUASrc = 1'bx;
                        ALUBSrc = 1'b1;
                        ALUOp = 4'b1111;
                        ImmSrc = 3'b010;
                        DMWr = 1'b0;
                        DMCtrl = 3'bxxx;
                        RUDataWrSrc = 2'b00;
                        RUWr = 1'b1;
                        BrOp = 5'b100xxx;
                end

                 7'b0110111: //Instruccion tipo U -auipc
                begin
                        ALUASrc = 1'b1;
                        ALUBSrc = 1'b1;
                        ALUOp = 4'b0000;
                        ImmSrc = 3'b010;
                        DMWr = 1'b0;
                        DMCtrl = 3'bxxx;
                        RUDataWrSrc = 2'b00;
                        RUWr = 1'b1;
                        BrOp = 5'b100xxx;
                end
          endcase
        end
  endmodule
```

Generador de inmediatos

El generador de inmediatos es el encargado de ensamblar el inmediato decodificado en la instrucción de entrada según el tipo de esta

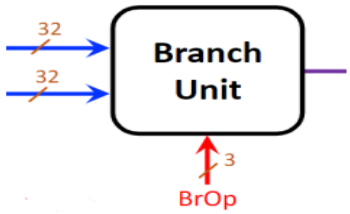



```
module imm_gen(
  input logic [31:0] Inst,
  input logic [2:0] ImmSrc,
  output logic [31:0] ImmExt);

  always @ (*)
    begin
      case (ImmSrc)
```

```
        3'b000: //Intruccion tipo I
           ImmExt = {{20{Inst{31}}}, Inst {31:20}};
        3'b001: //Intruccion tipo S
          ImmExt = {{20{Inst{31}}}, Inst {31:25}, Inst{11:7}};
        3'b101: //Intruccion tipo B
          ImmExt = {{19{Inst{31}}}, Inst {31}, Inst{7},Inst {30:25},Inst
   {11:8}, 1'b0};
        3'b010: //Intruccion tipo U
          ImmExt = {{Inst{Inst{31:12}}}, 12'b0};
        3'b110: //Intruccion tipo J
          ImmExt = {{19{Inst{31}}}, Inst {31}, Inst{19:12},Inst
   {20},Inst {30:21}, 1'b0};
            default;
            ImmExt = 31'bx;
      endcase
    end
endmodule
```

Branch unit

Esta se encarga de determinar si la instrucción de entrada ordena un salto en la dirección de la memoria de instrucciones




```
module branch_unit(
  input logic signed  [31:0] A,//Es asigando de RU [rs1]
  input logic signed  [31:0] B, //Es asignado de RU [rs2]
  input logic  [4:0] BrOp,
  output logic NextPCSrc );

  alwats @ (*)
       begin
         if (BrOp[4])
                NextPCSrc = 1;
              else
                if (BrOp [3])
                  case (BrO[2:0])
                    3'b000; //BEQ
                      NextPCSrc = (A == B);
                    3'b000; //BNE
                      NextPCSrc = (A != B);
                    3'b100; //BLT
                      NextPCSrc = (A < B);
                    3'b101; //BGE
                      NextPCSrc = (A >= B);
                    3'b110; //BLTU
```

```
                        NextPCSrc = ($unsigned (A) < $unsigned (B));
                        3'b111; //BGEU
                        NextPCSrc = ($unsigned (A)  >= $unsigned (B));
                        default:
                                NextPCSrc = 1'bx;
                    endcase
                else
                        NextPCSrc = 0;
        end
  endmodule

```

Program counter

```
module program_counter (
        input logic CLK,
        input logic [31:0] PC_In,
  output logic [31:0] PC_Out = 0 );

  always @ (posedge CLK)
    PC_Out <= PC_In;
endmodule
```

Adder

```
module adder (
  input [31:0] adder_In,
  output [31:0] adder_Out );

        assign adder_Out = adder_In +  4;
endmodule
```

MUX 2x1

```
module MUX2x1 (
  input logic [31:0]In1,
  input logic [31:0]In2,
  input logic  Src,
  output logic [31:0]Out);

  always @ (*)
         begin
           case (Src)
                  1'b0:
                         Out <= In1;
                  1'b1:
                         Out <= In2;
           endcase
         end
endmodule
```

MUX 3x1

```
module MUX3x1 (
  input logic [31:0]In1,
  input logic [31:0]In2,
  input logic [31:0]In3,
```

```
  input logic  [1:0]RUDataWrSrc,
  output logic [31:0]Out);

  always @ (*)
         begin
           case (RUDataWeSrc)
                  2'b00:
                         Out <= In1;
                  2'b01:
                         Out <= In2;
                  2'b10:
                         Out <= In3;
           endcase
         end
endmodule
```

Testbench

```
module testbench;
       logic CLK;

  procesador_monociclo pm (CLK)

        always
               #5
               CLK = ~CLK;
       initial
              begin
                $dumpfile ("dump.vcd");
                $dumpvars();
                     CLK=0;
                     #850
                     $finish;
              end
endmodule
```

Instructions

```
  FF010113
00812623
00912423
01212223
01312023
00050413
00000913
00000993
00000993
0099D863
00890933
00198993
FE000AE3
00090513
00C12403
00812483
00412903
00012983
```

```
01010113
00008067
00040513
0FC000EF
00050493
FF010113
00812623
00912423
01212223
00112023
00050413
00100913
00140413
00100493
0084DE63
00090513
00048593
DD1FF0EF
00050913
00148493
FE0004E3
00090513
00C12403
00812483
00412903
00012083
00012083
01010113
00008067
```

Registers

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Resultado

```
#KERNEL:  x20 value    0
#KERNEL:  x20 value    72
RUNTIME: Info:  RUNTIME_0068 testbemch.sv  (16):  $finish calle.
#KERNEL: Time: 1 us, Iteration:  0, Instance: / testbench, Process:
   @INITIAL#19_1@.
#KERNEL: stopped at time: 1 us
#VSIM: Simulation has finished. There are no more test vectors  to
   simulate.
#VSIM: Simulatiom has finished
```

Simulación

Instrucciones RV32I

Tipo R

	OpCode	Funct3	Funct7
add	0110011	000	0000000
sub	0110011	000	0100000
sll	0110011	001	0000000
slt	0110011	010	0000000
sltu	0110011	011	0000000
xor	0110011	100	0000000
srl	0110011	101	0000000
sra	0110011	101	0100000
or	0110011	110	0000000
and	0110011	111	0000000

Tipo I

	OpCode	Funct3
addi	0010011	000
slli	0010011	001
slti	0010011	010
sltiu	0010011	011
xori	0010011	100
srli	0010011	101
srai	0010011	101
ori	0010011	110
andi	0010011	111
lb	0000011	000
lh	0000011	001
lw	0000011	010
lbu	0000011	100
lhu	0000011	101
jalr	1100111	000

Tipo S B

	OpCode	Funct3
beq	1100011	000
bne	1100011	001
blt	1100011	100
bge	1100011	101
bltu	1100011	110
bgeu	1100011	111
sb	0100011	000
sh	0100011	001
sw	0100011	010

Tipo U J

	OpCode
jal	1101111
lui	0110111
auipc	0010111

Jose A. Jaramillo Villegas

GRUPO SIRIUS

Universidad Tecnológica de Pereira

Crear un código en C ++, llevar a ensamblador y por último a lenguaje de máquina. El cual será utilizado como las instrucciones de prueba para demostrar el correcto funcionamiento.

```
  Funcion en C++

o Funcion en C++

int multi(int x, int y)
{
int Acc = 0;
for(int i = 0;i < y;i++) Acc += x;
return Acc;
}

int factorial(int num)
{
int fact = 1;
int num += 1;
for(int i = 1;i < num;i++) fact = multi(fact,i); return fact;
}

o Funcion en lenguaje ensamblador

Prototipo de la funcion multi:
x10      x      x8 x11       y      x9
Acc      x18
i      x19 multi(x,y)      x10

Prototipo de la funcion factorial: x10       num      x8
i      x9
fact       x18 factorial(num)      x10
```

```
29
30 Llamado a la funci n factorial: f = factorial(num)
31 num      x8
32 f      x9
```

Lenguaje ensamblador			
Address	**Label**	**Instruction**	**Description**
0x0…4200	multi:	addi sp,sp,-16	**Reservamos espacio en la pila**
0x0…4204		sw x8,12(sp)	**Guardamos en los espacios reservados anteriormente**
0x0…4208		sw x9,8(sp)	
0x0…420C		sw x18,4(sp)	
0x0…4210		sw x19,0(sp)	
0x0…4214		addi x8,x10,0	**Movemos los argumentos**
0x0…4218		addi x9,x11,0	
0x0…421C		addi x18,x0,0	**Acc = 0**
0x0…4220		addi x19,x0,0	**i = 0**
0x0…4224	for0:	bge x19,x9,endfor0	**Cuando (i >= y) salir del for**
0x0…4228		add x18,x18,x8	**Acc += x**
0x0…422C		addi x19,x19,1	**i++**
0x0…4230		beq x0,x0,for0	
0x0…4234	endfor0:	addi x10,x18,0	**return Acc**
0x0…4238		lw x8,12(sp)	**Recuperar de la pila los valores en los que recibí los registros.**
0x0…423C		lw x9,8(sp)	
0x0…4240		lw x18,4(sp)	
0x0…4244		lw x19,0(sp)	
0x0…4248		addi sp,sp,16	**Liberar el espacio reservado**
0x0…424C		jalr x0,0(x1)	**Retorno**
…			
0x0…4300		addi x10,x8,0	
0x0…4304		jal x1,factorial	
0x0…4308		addi x9,x10,0	
…			
0x0…4400	factorial:	addi sp,sp,-16	**Reservar espacio en la pila.**
0x0…4404		sw x8,12(sp)	**Guardar en el espacio reservado en la pila los registros.**
0x0…4408		sw x9,8(sp)	
0x0…440C		sw x18,4(sp)	
0x0…4410		sw x1,0(sp)	
0x0…4414		addi x8,x10,0	**Mover los argumentos**
0x0…4418		addi x18,x0,1	**fact = 1**
0x0…441C		addi x8,x8,1	**num += 1**
0x0…4420		addi x9,x0,1	**i = 1**
0x0…4424	for0:	bge x9,x8,endfor0	**Si (i >= num) salir del for**
0x0…4428		addi x10,x18,0	**fact = multi(fact,i)**
0x0…442C		addi x11,x9,0	
0x0…4430		jal x1,multi	
0x0…4434		addi x18,x10,0	
0x0…4438		addi x9,x9,1	**i++**
0x0…443C		beq x0,x0,for0	

0x0…4440	endfor0:	addi x10,x18,0	**return fact**
0x0…4444		lw x8,12(sp)	**Recuperar de la pila los valores en los que recibí los registros.**
0x0…4448		lw x9,8(sp)	
0x0…444C		lw x18,4(sp)	
0x0…4450		lw x1,0(sp)	
0x0…4454		addi sp,sp,16	**Liberar el espacio reservado**
0x0…4458		jalr x0,0(x1)	**Retorno**

Conversión lenguaje de Máquina Función Multi

Addi sp,so,-16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
F				F				0				1				0				1				1				3			

Hexadecimal 0Xff010113

Sw x8, 12 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1
0				0			8					1				2				6				2				3			

Hexadecimal 0x00812623

Sw x9, 8 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1
0				0			9					1				2				4				2				3			

Hexadecimal 0x00912423

Sw x18, 4 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
0				1			2					1				2				2				2				3			

Hexadecimal 0x01212223

Sw x19, 0 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1
9				1			3					1				2				0				2				3			

Hexadecimal 0x1312023

Addi x8, x10, 0

Imm [11:5]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			0					5				0				4				1				3			

Hexadecimal 0x00050413

Addi x18,x0,0

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0			0					5				8				4				9				3			

Hexadecimal 0x0058493

Addi x19, x0,0

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1
0				0			0					0				0				9				9				3			

Hexadecimal 0x00000913

Bge x19,x9, end for ()

Imm [12,10;5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	1	1	0	0	0	1	1
0				0			9					9					D			8					6			3			

Hexadecimal 0x0099D863

Add, x18,x18,x8

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	1
0				0			8					9					0			9					3			3			

Hexadecimal 0x00890933

Addi x19, x19, 1

Imm [11:0]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1
0				0			1					9					8			9					3			3			

Hexadecimal 0x00198993

Beq x0, x0, for 0

Imm [12,10;15]							Rs2					Rs1					Funct3			Imm [4:0,11]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	1
F				E			0					0					0			A					E			3			

Hexadecimal 0XFE000AE3

Addi x10, x18,0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0			0					9					0			5					2			3			

Hexadecimal 0x00090513

lw x8, 12 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1
0				0			C					1					2			4					0			3			

Hexadecimal 0x00C12403

lw x9,8(sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	1
0				0			8					1					2			4					8			3			

Hexadecimal 0x00812483

lw x18, 4 (2p)

Imm [11:0]							Rs2					Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1
0				0			4					1					2			9					0			3			

Hexadecimal 0x00412903

Iw x19, 0 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0	0	1	1
0				0				0				1					2			9					8					3	

Hexadecimal 0x00012983

Addi sp, sp, 16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
0				1				0				1					0			1					1					3	

Hexadecimal 0x01010113

Jarlr x0, 0 (x1)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1
0				0				0				0					8			0					6					7	

Hexadecimal 0x00008067

Addi x10, x8,0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				4					0			5					1					3	

Hexadecimal 0x00040513

Jal x1, factorial

Imm [20,10: 1, 11,19:12]																				rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1
0				F				C				0					0			0					E					F	

Hexadecimal 0x0FC000EF

Addi x9, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0				0				5					0			4					9					3	

Hexadecimal 0x0050493

Addi sp, sp, -16

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
F				F				0				1					0			1					1					3	

Hexadecimal 0xFF010113

Sw x8, 12 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1
0				0				8				1					2			6					2					3	

Hexadecimal 0x00812623

Sw x9, 8 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1
0				0			9					1					2			4					2			3			

Hexadecimal 0x00912423

Sw x18, 4 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
0				1			2					1					2			2					2			3			

Hexadecimal 0x01212223

W2 x1, 0 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1
0				0			1					1					2			0					2			3			

Hexadecimal 0x00112023

Addi x8, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			0					5					0			4					1			3			

Hexadecimal 0x00050413

Addi x18, x0, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1
0				0			1					0					0			9					1			3			

Hexadecimal 0x00100913

Addi x8 , x8, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			1					4					0			4					1			3			

Hexadecimal 0x00140413

Addi x9, x0, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0			1					0								4					9			3			

Hexadecimal 0x00100493

Bge x9, x8, endfor 0

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	0	1	1
0				0			8					4					D			E					6			3			

Hexadecimal 0x0084DE63

Addi x10, x18, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				9				0				5				1				3			

Hexadecimal 0x00090513

Addi x11, x9, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	1	1
0				0				0				4				8				5				9				3			

Hexadecimal 0x00048593

Jal x, multi

Imm [11:5]																				rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	0	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	1	
D				D				1				F				F				0				E				F			

0xDDIFFOEF

Addi x18, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1
0				0				0				5				0				9				1				3			

Hexadecimal 0x00050913

Addi x9, x9, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0				1				4				8				4				9				3			

Hexadecimal 0x00148493

Beq x0, x0,for 0

Imm [11:0]							Rs2					Rs1					Funct3			Imm [4:0,11]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	1
F				E				0				0				0				4				E				3			

Hexadecimal 0XFE0004E3

Add x10, x18, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				9				0				5				1				3			

Hexadecimal 0x00090513

Iw x8, 12 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1

0	0	C	1	2	4	0	3

Hexadecimal 0x00C12403

lw x9, 8 (sp)

Imm [11:0]												Rs1					Funct3			Ird					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	1
0				0				8				1					2			4					8					3	

Hexadecimal 0x00812483

lw x18, 4 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1
0				0				4				1					2			9					0					3	

Hexadecimal 0x00812623

lw x1, 0 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1
0				0				0				1					2			0					8					3	

Hexadecimal 0x00812623

Addi sp, sp, 16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
0				1				0				1					0			1					1					3	

Hexadecimal 0x01010113

Jarl x0, 0 (x1)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1
0				0				0				8					0			6					7						

Hexadecimal 0x00008067

Procedimiento del laboratorio:

Proceso realizado para el diseño del ejercicio.
Para el diseño del procesador monociclo RISC-V se implementó teniendo en cuenta las instrucciones suministradas y componentes que fueron proporcionados en el gráfico de dicho procesador, primero se declaraban teniendo en cuenta y agregando como argumentos las variables de entrada y salida, luego se describía el funcionamiento interno de cada módulo.

A continuación se concetaron los modulos y se distanciaron los módulos de cada proceso, además de se asiganaron las instrucciones pertinentes para verificar el funcionamiento correcto de cada módulo y su conexión con módulos posteriores.

Problemas encontrados durante el diseño con sus respectivas las soluciones.

Se encontraron varios problemas en relación al distanciamiento de los módulos y la conexión entre varios componentes, así como algunas implementaciones en las cuales era necesario saber cierta sintaxis del lenguaje para describir el funcionamiento de dichos componentes, consultando en internet y entrando en foros se logró dar con la solución, además de la documentación que fue proporcionada en el Microsft Teams.

Interpretación de los resultados de la simulación.

En la simulación se puede observar la señal de cada uno de los componentes y como se conectaban entre si, para dar lugar al resultado de una operación matemática, se verificó el funcionamiento haciendo manualmente la intrucción y poteriormente se verificaron los resultados

Conclusione

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. El tipo de procesador más comúnmente utilizado en equipos de escritorio, el x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas traducen instrucciones basadas en CISC x86 a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

El objetivo de diseñar máquinas la arquitectura RISC FIVE es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC,2 DEC Alpha, MIPS, ARM, SPARC son ejemplos de algunos de ellos